LIS Developer's Guide

Submitted under Task Agreement GSFC-CT-2

Cooperative Agreement Notice (CAN) CAN-00OES-01

Increasing Interoperability and Performance of

Grand Challenge Applications in the Earth, Space, Life, and

Microgravity Sciences

December 2003

Version 2.3

History:

Revision	Summary of Changes	Date
2.3	LIS 2.3 code release	December 19, 2003

NASA Goddard Space Flight Center,
Greenbelt, MD 20771

Contents

1 Introduction **3**

2 Coding and Documentation Conventions **4**
2.1 Coding conventions 4
2.2 Documentation conventions 4

3 Customizable Features in LIS **7**
3.1 Function Tables 7

4 Customizing LIS to use new land surface models **9**

5 Customizing LIS to use new forcing schemes **13**

6 Customizing LIS for a new domain **16**

7 Customizing LIS for Input data **19**
7.1 Soils data 19
7.2 Vegetation data 19
7.3 Meterorological data 20

1 Introduction

The primary goal of the LIS project is to build a system that is capable of performing high resolution land surface modeling at high performance using scalable computing technologies. The LIS software system consists of a number of components: (1) LIS driver: the core software that integrates the use of land surface models, data management techniques, and high performance computing. (2) community land surface models such as CLM [2], Noah [3], and VIC [5], and (3) Visualization tools. One of the important design goals of LIS is to develop an interoperable system to interface and interoperate with land surface modeling community and other earth system models. LIS is designed using an object oriented, componet-based style. The adaptable interfaces in LIS can be used by the developers to ease the cost of development and foster rapid prototyping and development of applications.

This document describes some of the interoperable features in LIS and how to use/extend them. The following sections describe the general development and documentation practices recommended for using and extending LIS software, followed by the guidelines for using the extensible features in LIS for customization and improved functionality.

2 Coding and Documentation Conventions

This section describes some of the coding and documentation conventions [1] that are helpful for developers of LIS.

2.1 Coding conventions

LIS considers F90 and C as implementation languages. Different compilers often employ various levels of strictness in parsing source files based on extensions such as f77, F, f, and f90. This makes the task of porting code to different platforms a hard process. Therefore, fortran additions and contributions to LIS code are expected to be in F90 format. Some of the style guidelines followed in LIS are as follows:

- Preprocessor: C preprocessor (cpp) is used where ever the use of a language preprocessor is required. The fortran compiler is assumed to have the ability to run the preprocessor as part of the compilation process. The preprocessing tokens are written in uppercase to distinguish them from the Fortran code.
- Loops: All fortran loops in fortran are structured using do-enddo constructs as opposed to numbered loops.
- Indentation: Code with nested if blocks and do loops are expected to be indented for readability.
- Modules: Modules must be named the same as the file in which they reside. This is enforced due to the fact that make programs build dependencies based on file names.
- Implicit none: All variables in different modules should be explicitly typed by the use of "implicit none" statement.

2.2 Documentation conventions

LIS uses an in-line documentation system that allows users to creat both web-browsable (html) and print-friendly(ps/pdf) documentation. Each function, subroutine, or module includes a prologue instrumented for use with the ProTex auto-documentation script [4]. The following examples describe the documentation templates used in LIS.

Templates for routines that are not internal to modules.

```
!-------------------------------------------------------------------------
!          NASA/GSFC Land Information Systems LIS 2.3
!-------------------------------------------------------------------------
!BOP
!
! !ROUTINE:
```

```
!
! !INTERFACE:
!
! !USES:
!
! !INPUT PARAMETERS:
!
! !OUTPUT PARAMETERS:
!
! !DESCRIPTION:
!
! !BUGS:
!
! !SEE ALSO:
!
! !SYSTEM ROUTINES:
!
! !FILES USED:
!
! !REVISION HISTORY:
!
!  27Jun02  Username Initial specification
!
!EOP
!-------------------------------------------------------------------------
!BOC
!EOC
```

Template for a module :

```
!-------------------------------------------------------------------------
!         NASA/GSFC Land Information Systems LIS 2.3
!-------------------------------------------------------------------------
!BOP
!
! !MODULE:
!
! !PUBLIC TYPES:
!
! !PUBLIC MEMBER FUNCTIONS:
!
! !PUBLIC DATA MEMBERS:
!
! !DESCRIPTION:
!
! !REVISION HISTORY:
```

```
!
!  27Jun02  Username Initial specification
!
!EOP
```

Template for a C file:

```
//-------------------------------------------------------------------------
//         NASA/GSFC Land Information Systems LIS 2.3
//-------------------------------------------------------------------------
//BOP
//
// !ROUTINE:
//
// !INTERFACE:
//
// !USES:
//
// !INPUT PARAMETERS:
//
// !OUTPUT PARAMETERS:
//
// !DESCRIPTION:
//
// !BUGS:
//
// !SEE ALSO:
//
// !SYSTEM ROUTINES:
//
// !FILES USED:
//
// !REVISION HISTORY:
//
//  27Jun02  Username Initial specification
//
//EOP
//-------------------------------------------------------------------------
//BOC
//EOC
```

3 Customizable Features in LIS

The LIS driver is designed with extensible interfaces for facilitating easy incorporation of new features into LIS. The LIS driver uses advanced features of Fortran 90 programming language, which are especially suitable for object oriented programming. The object oriented style of design adopted in LIS enables the driver to provide well defined interfaces or "plug points" for enabling rapid prototyping and development of new features and applications into LIS.

The LIS driver includes a number of functional extensions including:

- land surface model: Interfaces for adding new land surface models.
- base forcing: Interfaces for adding new model forcing schemes.
- observed forcing: Interfaces for adding observational forcing products.
- domain: Using a user specified domain or a subdomain of interest.
- parameters: Specifying custom defined datasets.

3.1 Function Tables

The modules in LIS are constructed using a component-based design, with each module/component representing a program segment that is functionally related. The customizable interfaces in LIS are designed using a number of virtual function tables and the actual delegation of the calls are done at runtime by resolving the function names from the table. C language allows the capability to store functions, table them, and pass them as arguments. F90 allows passing of functions as arguments. By combining these features of both languages, LIS uses a complete set of operations with function pointers.

Figure 1 illustrates how the function tables work in LIS. A function is stored in the table typically by a `register` function, that simply stores the pointer to the function at the specified index. When the function needs to be accessed, a generic call is made which resolves into a specific call depending on the index specified. This type of implementation helps in defining generic calls in the driver and to include only the components of interest while compiling and building the executable. For simplicity, throughout this document the word "registry" is used to refer to a function table.

The LIS.2.3 source code available from the LIS website contains a number of subdirectories, which are organized as components. The top level organization of the source (*src*) is as follows:

Directory Name	Synopsis
driver	LIS driver routines
baseforcing	Routines to call model forcing methods
obsprecips	Routines to call observed precipitation products
obsrads	Routines to call observed radiation products
forcing-plugin	Routines that define registries for forcing schemes
lsm-plugin	Routines that defines registries for land surface models
lsms	Contains land surface model codes

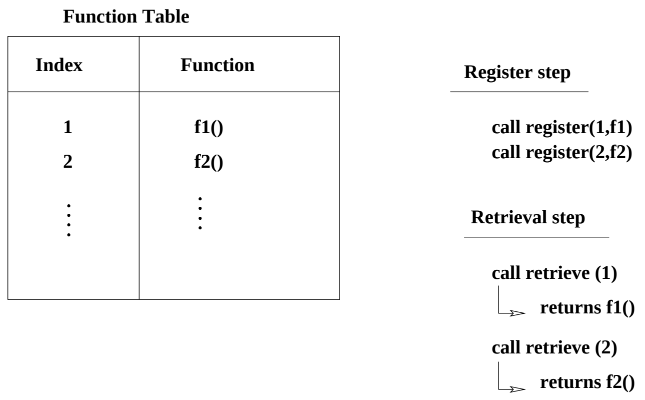

Figure 1: Example of a function table implementation

4 Customizing LIS to use new land surface models

The *lsm-plugin* directory contains the *lsm_pluginMod* module that can be used to customize and define land surface models in LIS. The *lsm_pluginMod* contains a *lsm_plugin* method that defines a number or registries to capture the basic offline operations of a land surface model. The registries can be used to define functions to perform the following tasks:

- initialization
  Definition of land surface model variables, allocation of memory, reading runtime parameters, etc.
- setup
  Initialization of land surface model parameters.
- dynamic setup
  Routines to initialize or update time dependent parameters.
- run
  Routines to execute land surface model on a single gridcell for a single timestep.
- write restart
  Routines to write restart files
- read restart
  Routines to read restart files
- output
  Routines to write output
- transfer of forcing data to model tiles
  Routines that provides an array of forcing variables for each gricell.

The following example shows how the registry functions are defined for Noah land surface model.

```
call registerlsmini(1,noah_varder_ini)
call registerlsmsetup(1,noah_setup)
call registerlsmdynsetup(1,noah_dynsetup)
call registerlsmrun(1,noah_main)
call registerlsmrestart(1,noahrst)
call registerlsmoutput(1,noah_output)
call registerlsmf2t(1,noah_f2t)
call registerlsmwrst(1,noah_writerst)
```

The registry functions defined for noah are:

`noah_varder_ini`	Initialization for Noah
`noah_setup`	Sets up Noah's parameters
`noah_dynsetup`	Sets up Noah's time dependant parameters
`noah_main`	Runs the Noah model on a single gridcell at a timestep
`noahrst`	Reads the Noah restart files
`noah_output`	Writes output of Noah runs
`noah_writerestart`	Writes Noah's restart files
`noah_f2t`	Transfers forcing data to Noah model tiles

The index used for Noah's functions in this case is 1. The index needs to be the same for all registry defintions for a particular model. The user, however, can define any integer value as the index chosen for a land surface model. The corresponding index needs to be specified in the lis card file (`LIS%d%LSM`) i.e., if Noah model is used, `LIS%d%LSM` should be assigned a value of 1.

The following code segment shows an example of defining two different land models (CLM and Noah) to be included in the LIS executable. The same procedure can be extended to define more models, or customize LIS to use only the models of interest.

```
        call registerlsmini(1,noah_varder_ini)
        call registerlsmini(2,clm_varder_ini)

        call registerlsmsetup(1,noah_setup)
        call registerlsmsetup(2,clm2_setup)

        call registerlsmdynsetup(1,noah_dynsetup)
        call registerlsmdynsetup(2,clm2_dynsetup)

        call registerlsmrun(1,noah_main)
        call registerlsmrun(2,driver)

        call registerlsmrestart(1,noahrst)
        call registerlsmrestart(2,clm2_restart)

        call registerlsmoutput(1,noah_output)
        call registerlsmoutput(2,clm2_output)

        call registerlsmf2t(1,noah_f2t)
        call registerlsmf2t(2,atmdrv)

        call registerlsmwrst(1,noah_writerst)
        call registerlsmwrst(2,clm2wrst)
```

The runtime specific parameters for a land surface model can be read in at runtime through the lis card file. The user needs to specify a customized namelist and provide routines for reading the same. For Noah runs, the section of the lis card file contains a namelist segment such as:

```
&noah
noahdrv%WRITEINTN    = 3
noahdrv%NOAH_RFILE   = "noah.rst"
noahdrv%NOAH_MGFILE  = "/X6RAID/MODIS-0.25/"
noahdrv%NOAH_ALBFILE = "/X6RAID/MODIS-0.25/"
noahdrv%NOAH_VFILE   = "BCS/noah_parms/noah.vegparms.txt"
noahdrv%NOAH_SFILE   = "BCS/noah_parms/noah.soilparms.txt"
noahdrv%NOAH_MXSNAL  = "/X6RAID/MODIS-0.25/maxsnalb.bfsa"
noahdrv%NOAH_TBOT    = "/X6RAID/MODIS-0.25/tbot.bfsa"
noahdrv%NOAH_ISM     = 0.30
noahdrv%NOAH_IT      = 290.0
noahdrv%NOAH_NVEGP   = 7
noahdrv%NOAH_NSOILP  = 10
/
```

The namelist specifies variables such as locations of parameter files, output writing intervals, initial conditions, etc. The routine to read these variables are typically done during initialization of the land surface model. The program segment for Noah is shown below as an example. For explanation of other routines, please refer to the source code documentation.

```
!---------------------------------------------------------------
! Reads the Noah name list
!---------------------------------------------------------------
  if(masterproc) then
       call readnoahcrd(noahdrv)
    endif
!---------------------------------------------------------------
! Defines the derived types for MPI, and broadcasts the
! namelist variables to all processors. This step can be
! skipped if using a sequential execution.
!---------------------------------------------------------------
    call def_noahpar_struct
    call MPI_BCAST(noahdrv, 1, MPI_NOAHDRV_STRUCT, 0, &
         MPI_COMM_WORLD, ierr)
!--------------------------------------------------------------
! Allocates Memory for Noah variables
!--------------------------------------------------------------
    if(masterproc) then
       allocate(noah(nch))
    else
       allocate(noah(di_array(iam)))
    endif
```

The *src/driver* directory contains a number of modules that provides helpful variables that may be required while defining land surface model specific routines. Some of the useful modules and the variable provided by them are listed below. For more details, please refer to the source code documentation.

Module name	Functions
time_manager	Variables and routines for time management
lisdrf_module	`grid` : Vector representation of the running domain grid
	`tile` : Vector representation of the running domain tiles
	`gindex` : Mapping of the running domain grid to the corresponding 2-D grid
grid_spmdMod	Variables and routines that define domain decomposition of the vector grid space
tile_spmdMod	Variables and routines that define domain decomposition of the vector tile space
def_ipMod	Variables and routines that are required to carry out spatial interpolation of scalar data

5 Customizing LIS to use new forcing schemes

The boundary conditions describing the (upper) atmospheric fluxes are known as "forcings". LIS makes use of model derived data as well as satellite and ground-based observational data as forcings. The land surface models are typically run using model derived data. The observational data is used to overwrite the model derived data, whenever they are available. LIS driver provides interfaces to incorporate model derived (base) forcing schemes as well as observational (currently for radiation and precipitation products) forcing schemes.

The *forcing-plugin* directory contains modules *baseforcing_pluginMod*, *precipforcing_pluginMod*, and *radfocing_pluginMod*, that can be used to customize and define base forcing schemes, observed precipitation forcing schemes, and observed radiation forcing schemes, respectively. These modules provide plugin routines *baseforcing_plugin*, *precipforcing_plugin*, and *radforcing_plugin*, respectively.

baseforcing_module provides registries to define functions to perform the following tasks.

- definition of native domain
  Routines to define the native domain of the forcing data, read runtime specific parameters through a namelist, etc.
- retrieval of forcing data
  Routines to retrieve the forcing data, and interpolate them.
- temporal interpolation
  Routines to interpolate data temporally.

The following code segment shows how two baseforcing schemes are included in LIS.

```
call registerdefnat(1,defnatgdas)
call registerdefnat(2,defnatgeos)

call registerget(1,getgdas)
call registerget(2,getgeos)

call registertimeinterp(1,time_interp_gdas)
call registertimeinterp(2,time_interp_geos)
```

Similar to the case in *lsm_pluginMod*, the indices used in the registries need to be consistent for a particular scheme. In the example shown above, GDAS forcing scheme is assigned index 1 and GEOS scheme is assigned 2. These indices are arbitrary, but the indices used in the card file (`LIS%d%FORCE`) should reflect the ones defined in the registry. i.e., if GEOS forcing scheme is to be used, `LIS%d%FORCE` should be assigned a value of 2.

The runtime specific parameters for a forcing scheme can be specified at runtime through the lis card file. The user needs to specify a customized namelist and provide routines for reading the same. For GEOS runs, the section of the lis card file contains a namelist segment such as:

```
&geos
geosdrv%GEOSDIR    = "/X6RAID/DATA/GEOS/BEST_LK"
geosdrv%NROLD      = 181
geosdrv%NCOLD      = 360
geosdrv%NMIF       = 13
/
```

The namelist specifies variables such as locations of forcings files, the native domain sizes, the number of variables in each file, etc. The routine to read these parameter is done typically while defining the native domain parameters of the forcing scheme. A sample routine for GEOS forcing scheme is shown below.

```
!-------------------------------------------------------------
! Reads the GEOS name list
!-------------------------------------------------------------
    call readgeoscrd(geosdrv)
!-------------------------------------------------------------
! Defines the native GEOS domain as a kgds array
!-------------------------------------------------------------
    kgdsi(1) = 0
    kgdsi(2) = geosdrv%ncold
    kgdsi(3) = geosdrv%nrold
    kgdsi(4) = -90000
    kgdsi(7) = 90000
    kgdsi(5) = -180000
    kgdsi(6) = 128
    kgdsi(8) = 179000
    kgdsi(9) = 1000
    kgdsi(10) = 1000
    kgdsi(20) = 255
    mi = geosdrv%ncold*geosdrv%nrold
```

precipforcing_module provides registries to define functions to perform the following tasks.

- definition of native domain
  Routines to define the native domain of the forcing data, read runtime specific parameters through a namelist, etc.
- retrieval of forcing data
  Routines to retrieve forcing data and interpolate them.

The following code segment shows how the CMAP precipitation scheme is included in LIS.

```
    call registerdefnatpcp(4,defnatcmap)
    call registerpget(4,getcmap)
```

The indexing scheme is similar to the cases described above. In this case, the CMAP scheme is assigned an index of 4. `LIS%f%GPCPSRC` in the card file should correspond to the indices defined in the registries. A value of 0 indicates that no observed precipitation scheme will be employed.

The customized namelist section for CMAP is shown below.

```
&cmap
cmapdrv%CMAPDIR     = "./input/CMAP"
cmapdrv%NROLD       = 181
cmapdrv%NCOLD       = 360
/
```

The namelist specifies variables such as locations of forcings files, the native domain sizes, etc. The routine to read these parameter is done typically while defining the native domain parameters, similar to the base forcing case.

The design of *radforcing_module* is similar to the cases described above. The registry functions for this module are:

- definition of native domain
  Routines to define the native domain.
- retrieval of forcing data
  Routines to retrieve and interpolate data.
- Interpolate data in time
  Temporallly interpolate data.

An example of using AGRMET observed radiation scheme is shown below.

```
    call registerrget(1,getgrad)
    call registerdefnatrad(1,defnatagrmet)
    call registerrti(1,time_interp_agrmet)
```

The indices defined for observed radiation schemes correspond to the `LIS%f%RADSRC` in the card file. The value is defined to be 0 if no observed radiation schemes are used.

As mentioned earlier, the modules in *src/driver* can be used in defining routines needed for defining a forcing scheme in LIS. Please refer to the source code documentation for details.

6 Customizing LIS for a new domain

The LIS driver is designed to be domain independent. The parameters used to define the domain are designed to be runtime options. The user is required to specify two different types of domain information

- Running domain
  The domain over which land surface simulations are carried out.
- Data domain
  The domain over which data sets are defined. Currently it is assumed that all data sets are defined in the same domain. In future, support for defining domain information for each dataset will be provided.

The parameters used to define a domain are adopted from the design used in grib decoding programs such as w3fi63 [6]. The *domain* namelist in the lis card file specifies an array called `kgds`. Indices from 0 to 20 in the `kgds` array define the running domain, and indices from 41 onwards define the data domain. The description of `kgds` array are shown below for the running domain. The data domain can be defined in a similar way. The following sections describe the `kgds` array used in the card file for the running domain.

Variable	Description
`LIS%d%kgds(1)`	0 - Latitude/Longitude
	1 - Mercator cylindrical
	3 - Lambert conformal conical
	4 - Gaussian cylindrical
	5 - Polar stereographic azimuthal
For latitude, longitude grids,	
`LIS%d%kgds(2)`	Number of points on a latitude circle
`LIS%d%kgds(3)`	Number of points on a longitude circle
`LIS%d%kgds(4)`	Latitude of origin
`LIS%d%kgds(5)`	Longitude of origin
`LIS%d%kgds(6)`	Resolution flag
`LIS%d%kgds(7)`	Latitude of extreme point
`LIS%d%kgds(8)`	Longitude of extreme point
`LIS%d%kgds(9)`	Latitudinal directional increment
`LIS%d%kgds(10)`	Longitudinal directional increment
`LIS%d%kgds(11)`	Scanning mode flag

For Mercator grids,

`LIS%d%kgds(2)`	Number of points on a latitude circle
`LIS%d%kgds(3)`	Number of points on a longitude circle
`LIS%d%kgds(4)`	Latitude of origin
`LIS%d%kgds(5)`	Longitude of origin
`LIS%d%kgds(6)`	Resolution flag
`LIS%d%kgds(7)`	Latitude of extreme point
`LIS%d%kgds(8)`	Longitude of extreme point
`LIS%d%kgds(9)`	Latitude of projection intersection
`LIS%d%kgds(10)`	Reserved
`LIS%d%kgds(11)`	Scanning mode flag
`LIS%d%kgds(12)`	Longditudinal directional increment
`LIS%d%kgds(13)`	Latitudinal directional increment

For Lambert conformal grids,

`LIS%d%kgds(2)`	Number of points along x-axis
`LIS%d%kgds(3)`	Number of points along y-axis
`LIS%d%kgds(4)`	Latitude of origin
`LIS%d%kgds(5)`	Longitude of origin
`LIS%d%kgds(6)`	Resolution flag
`LIS%d%kgds(7)`	Orientation of grid
`LIS%d%kgds(8)`	x-direction increment
`LIS%d%kgds(9)`	y-direction increment
`LIS%d%kgds(10)`	projection center flag
`LIS%d%kgds(11)`	Scanning mode flag
`LIS%d%kgds(12)`	First lat from pole of secant cone inter
`LIS%d%kgds(13)`	Second lat from pole of secant cone inter

For Gaussian grids,

`LIS%d%kgds(2)`	Number of points on a latitude circle
`LIS%d%kgds(3)`	Number of points on a longitude circle
`LIS%d%kgds(4)`	Latitude of origin
`LIS%d%kgds(5)`	Longitude of origin
`LIS%d%kgds(6)`	Resolution flag
`LIS%d%kgds(7)`	Latitude of extreme point
`LIS%d%kgds(8)`	Longitude of extreme point
`LIS%d%kgds(9)`	Latitudinal direction of increment
`LIS%d%kgds(10)`	Number of circles from pole to equator
`LIS%d%kgds(11)`	Scanning mode flag
`LIS%d%kgds(12)`	Number of vertical coord parameters
`LIS%d%kgds(13)`	Octet number of list of vert. coord. parameters or location of the list of number of points in each row (if no vert. coords are present) 255 if neither are present

For polar stereographic grids,

`LIS%d%kgds(2)`	Number of points on a latitude circle
`LIS%d%kgds(3)`	Number of points on a longitude circle
`LIS%d%kgds(4)`	Latitude of origin
`LIS%d%kgds(5)`	Longitude of origin
`LIS%d%kgds(6)`	Resolution flag
`LIS%d%kgds(7)`	Grid orientation
`LIS%d%kgds(8)`	x direction increment
`LIS%d%kgds(9)`	y direction increment
`LIS%d%kgds(10)`	projection center flag
`LIS%d%kgds(11)`	Scanning mode flag

The LIS driver currently supports latitude/longitude and gaussian grids. The support for other types of grids are in development. The user is expected to provide parameter data that is consistent with the domain specified at runtime. The details of specifying parameter data is explained in the next section.

7 Customizing LIS for Input data

LIS allows the user to specify a number of input data sets at runtime through the card file. These data sets are expected to be consistent with the type of domain and resolution specified. Land model specific parameter data are expected to be specified in the namelists specific to each land model. The input data for models in LIS are divided into three categories

- Soils data : static
- Vegetation data : some static and some dynamic
- Meterological data : at different frequencies

The soil and vegetation data are used to specify the characteristics of the land surface and the meterological data to provide forcing at the upper boundary of the land surface. A detailed description of parameter data used in LIS are available at `http://lis.gsfc.nasa.gov/Data/index.shtml`.

7.1 Soils data

LIS provides a number of overlapping data sets for specifying soil hydraulic properties: sand, clay, silt, and organic texture fractions, porosity maps, etc.

bfsa - binary, sequential access,
txt - text
bfda - binary, direct acess

Data file	Description	Format
`LIS%p%SAFILE`	Sand fraction map file	bfsa
`LIS%p%CLFILE`	Clay fraction map file	bfsa
`LIS%p%ISCFILE`	Soil color map file	bfsa
`LIS%p%ELEVFILE`	Elevation difference file	bfsa

Some of the model specific parameter data are specified below:

`noahdrv%noah_sfile`	Noah soil parameter file	txt
`vicdrv%vic_sfile`	VIC soil parameter file	txt

7.2 Vegetation data

LIS uses a number of files to specify both static and time-varying vegetation properties. Some of the files are :

`LIS%p%MFILE`	Land/Water map file for modeling	bfsa
`LIS%p%VFILE`	Vegetation classification map file	bfsa
`LIS%p%AVHRRDIR`	Location of AVHRR-based LAI/SAI files	bfda
`LIS%p%MODISDIR`	Location of MODIS-based LAI/SAI files	bfda
CLM		
`clmdrv%clm2_vfile`	CLM mapping from UMD to plant functional types	txt
Noah		
`noahdrv%noah_mgfile`	Location of monthly veg. greenness fraction	bfsa
`noahdrv%noah_albfile`	Location of quarterly snow free albedo	bfsa
`noahdrv%noah_vfile`	Noah static vegetation parameter file	txt
`noahdrv%noah_mxsnal`	Maximum snow free albedo	bfsa
`noahdrv%noah_tbot`	Bottom temperature	bfsa
VIC		
`vicdrv%vic_veglibfile`	VIC vegetation parameter file	txt

7.3 Meterorological data

LIS includes a number of forcing schemes, both model-derived as well as observation based. A summary of the forcing data schemes implemented in LIS are shown below.

Forcing scheme	Type	Frequency
GEOS	model derived	3 hourly
GDAS	model derived	3 hourly
AGRMET	observational (radiation)	hourly
CMAP	observational (precipitation)	3 hourly

Implementation of other forcing schemes are currently under development.

References

[1] Community climate system model, software developers guide. http://www.ccsm.ucar.edu/csm/working_groups/Software/dev_guide/dev_guide/.

[2] Community land model. http://www.cgd.ucar.edu/tss/clm/.

[3] Noah land surface model. http://www.emc.ncep.noaa.gov/mmb/gcp/noahlsm/README_2.2.htm.

[4] Protex documentation system. http://gmao.gsfc.nasa.gov/software/protex.

[5] Variable infiltration capacity (vic) model. http://www.hydro.washington.edu/Lettenmaier/Models/VIC/VIChome.html.

[6] W3fi63 program. http://dss.ucar.edu/datasets/ds609.1/software/mords/w3fi63.f.